Practical - 3

Write code to simulate requests coming from clients and distribute them among the servers using load balancing algorithms.

→ what is load balancing?
- Load balancing is the process of distributing incoming network traffic across multiple servers.
- Purpose:-
  - No single servers get overloaded.
  - All servers are used efficiently.
  - Systems stay fast, reliable and scalable

→ Load balancing algorithms:

1) Round Robin:- Each server gets a turn in strict ~~or~~ order.
- Maintains a list of ~~av~~ available server.
- Assign a ~~a~~ new request to next server in the list ~~amo~~ and move that server to the end of the list.
- This ensures that each server receives requests in a round - robin fashion.
- Basic web applications.
- Drawback: Even if server is already busy, it still gets new request.

classmate
Date
Page

2) Least Connections:
- Server with the fewest active client gets ~~out~~ next request.
- Maintain a counter for the number of active connections on each server.
- This distributes the ~~workload~~ requests based on current work loads.
- Video streaming services, chat server where processing time can vary a lot
- Drawbacks: Can cause overhead ∵ needs to continuously monitor active connections for all servers. Slower if connections are constantly switching.

3) Weighted round robin:
- Server with higher weight gets more request.
- Assign weights ~~to~~ each server based on processing power, memory or other relevant factors.
- In a cloud environment with mixed server size, if some server ~~has~~ are more powerful (more RAM, CPU) they are assigned more work.
- Draw back: Fixed weight can't handle changing server performance.

→ ~~sep~~

→ Real Life applications:-
- Google, facebook, Youtube: have millions of users, at one requests distributed across thousands

of servers.
- Banking applications:- checking balance, removing cash etc.
- Cloud systems (AWS, Azure).
- Gaming servers (PUBG, Fortnite).

→ Components in code:

1) <u>ClientRequest class:-</u>
- Simulates a user request comming to the system.
- Each request has:
  - request_id.
  - arrival_time.
  - processing_time.

2) <u>Server Class:-</u>
- Simulates a server that can handle client requests.
- Each server keeps track of
  - Active connections → how many clients it is currently serving.
  - Total requests handled.
  - Weight.
- When a server handes a request it:
  - ~~Increments its active connections.~~
  - Sleeps (waits) for the processing time.
  - ~~Decrements active~~ connections ~~after completing.~~

~~Trac~~ : Tracking of active connection is not needed but its used for future upgrade or changes to least connections.

~~- Its harmless overhead - helps in statistics, monitoring.~~

3) Class LoadBalancer

☆ We are using weighted round robin.

⋮

~~→ Multithreading:-~~

→ Multithreading:- Each request is handled on a new thread → simulating multiple users interacting with the system at the same time.

→ Locking (self. lock) : To prevent race conditions when multiple thread try to change data at ~~the~~ same time.

→ Randomness: Client arrival times and processing times are random.